Regression Analysis Using StatsModels

CS3753 Data Science

Prof. Weining Zhang

Regression Analysis

Suppose we want to know how to improve profit by adjusting investment on advertising, special feature design, coffee supply, etc.

From the data we have, we want find out

- Is there a relationship between ad x_1, feature design x_2, coffee supply x_3 and sales y?
- How strong is the relationship?
- What is the effect of x_1, x_2 and x_3?
- How strong is each of x_1, x_2, and x_3 related to y?

Here, x_i is a feature or predictor variable, y is response or dependent variable

In statistical modelling, regression analysis is a process to estimate the relationship between a dependent variable and a set of predictor variables. Typically, we estimate the the conditional expectation of the dependent variable given the predictor variables.

- If the dependent variable is continuous, a linear regression can be used
- If the dependent variable is binary, a logistic regression will be used

In Python, we have several ways to perform a regression analysis.

Topics

- Review Linear Regression
- Regression models proviced by the Statsmodels package

In [ ]:
```
from scipy import stats
import matplotlib.pyplot as plt
import numpy as np
from numpy import random
import pandas as pd
%pylab inline
```

Review of Linear Statistical Model

- The Linear Statistical Model assumes that Y and $x_1, x_2, \dots, x_k$ has the following relationship.
$$Y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon$$
    - where x_i are independent (meaning x_i is not a linear combination of other x_j's), ϵ is a random error with $E(\epsilon) = 0$, and β_i's are unknown coefficients.
- We want to find β_i's, so that
$$E(Y) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$
- Given a set of data as a sample, the task is to estimate β_i from the sample, so that
$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_k x_k$$

Methods to Fit Data to a Simple Linear Model

- A simple linear model is
$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x$$
- Two methods for simple linear regression are
    - Least Squares
$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$
    where $S_{xy} = \sum_{i=1}^{n}(x_1 - \bar{x})(y_i - \bar{y})$ and $S_{xx} = \sum_{i=1}^{n}(x_1 - \bar{x})^2$
    - System Equations
$$\hat{B} = (X'X)^{-1}X'Y$$
    where X is the matrix of values <1, x>

Example: Simple Linear Regression

Find $\hat{\beta}_1, \hat{\beta}_0$ for a linear model $\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$, using sample:

```
(X, Y) = {(-2, 0), (-1, 0), (0, 1), (1, 1), (2, 3)}
```

- Method of Least Squares. Using numpy package, we can calculate the solution.

```
In [ ]: x = np.array([-2, -1, 0, 1, 2])
        y = np.array([0, 0, 1, 1, 3])
        xBar = x.mean()
        yBar = y.mean()
        Sxy = ((x-xBar)*(y-yBar)).sum()
        Sxx = ((x-xBar)**2).sum()
        B_1 = Sxy/Sxx
        B_0 = yBar-B_1*xBar
        print("YHat = ", B_0, " + ", B_1, "x" )
```

- Method of System Equation. Again, we can use numpy to calculate the matrix solution.

In [ ]:
```
# Using matrix operations
X =np.array([[1, -2], [1, -1], [1, 0], [1, 1], [1, 2]])
Y = np.array([[0], [0], [1], [1], [3]])
B = np.linalg.inv(dot(X.T, X)).dot(dot(X.T, Y))
print(B)
print("YHat = ", B[0, 0], " + ", B[1, 0], "x" )
```

- Can also use numpy stats.linregress()

In [ ]:
```
# Alternatively, use the NumPy stats linear regeretion function
x = np.array([-2, -1, 0, 1, 2])
y = np.array([0, 0, 1, 1, 3])
b1, b0, r, p_val, stderr = stats.linregress(x,y)
print("YBar = ", b0, " + ", b1, "x")
```

Measure the Quality of Learned Model

Several performance metrics have been defined to measure the quality of regression models.

- T-test is used to measure statistical significance of individual x_i, with $H_0 : \beta_i = 0$.
- F-test is used to measure the statistical significance of $x_1, \dots, x_k$, with $H_0 : \beta_1 = \beta_2 = \cdots = \beta_k = 0$
- R^2 is used to measure proportion of variance in data that is explained by the model

Statsmodels

The Python Statsmodels (http://www.statsmodels.org/stable/index.html) module provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration. An extensive list of result statistics are available for each estimator.

- Linear Regression
- Generalized Linear Models
- Regression with Discrete Dependent Variable
- ANOVA: Analysis of Variance

Linear Regression

Linear models with independently and identically distributed errors, and for errors with heteroscedasticity or autocorrelation. This module allows estimation by

- ordinary least squares (OLS),
- weighted least squares (WLS),
- generalized least squares (GLS), and
- feasible generalized least squares with autocorrelated AR(p) errors.

In [ ]:
```
%matplotlib inline

from __future__ import print_function
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std

np.random.seed(9876789)
```

Ordinary Least Squares (OLS)

- Prepare data for learning
- Instantiate an OLS model
- Fit the model
- Understand the summary of the model

Example: OLS for a Multi-linear Regression

We assume a dependent variable y and one independent variable x, and assume the linear model is

$$y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \epsilon$$

- Generate a data set

  For the data set, we will generate 100 data points within the range of $[0, 10]$, and add a standard normal distributed error

In [ ]:
```
nsample = 100
x = np.linspace(0, 10, 100) # x is uniformly placed 100 points
                            # between 0 and 10
X = np.column_stack((x, x**2)) # values of x_1 and x_2
beta = np.array([1, 0.1, 10]) # values of B_0, B_1 B_2
e = np.random.normal(size=nsample) # random errors
X[:10]
```

Our model needs an intercept so we add a column of 1s:

$$Y = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \epsilon$$

Here, β_0 is the intercept, or the value of Y when X=0. Notice that $x_1 = x, x_2 = x^2$

In [ ]:
```
X = sm.add_constant(X)
X[:10]
```

We then generate the set of y's.

In [ ]:
```
y = np.dot(X, beta) + e
y[:10]
```

The first two points:

- Instantiate and fit an OLS model, and print the regression result.

In [ ]:
```
model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

Understand the OLS Summary

- R-square and adj. R-square: proportion of variance in data that is explained by the model, we want 1.0
- F and Prob(F): statistical significance of all x_i to Y
- coef: type and strength of telationship between x_i and Y
- t and prob(t): Statistical significance of each x_i for Y
- p-value: Prob for $\beta_i = 0$, we want p-value<0.05
- [.025 .975]: 95% confidence interval of β_i, 95% chance true β_i is in the interval
- Skew: data symmetry. We want zero,
- Condition Number: the sensitivity of Y as compared to x_i's. We want 30<cond.N

Reference: https://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/ (https://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/)

Extract Performance Measure

Quantities of interest can be extracted directly from the fitted model.

```
dir(results): see thea full list result attributes
results.params: the coefficients
```

Here are some examples:

In [ ]:
```
dir(results)
```

In [ ]:
```
b = results.params
betas = "b0={0:4.3f}, b1={1:4.3f}, b2={2:4.3f}"
print('Parameters: ', betas.format(b[0], b[1], b[2]))
print("p-values: ", results.pvalues)
print('R2: ', results.rsquared)
message = "Fitted model: Y = {0:4.3f} + {1:4.3f} x1 + {2:4.3f} x2"
print(message.format(b[0], b[1], b[2]))
```

Example: OLS Non-linear Curve but Linear in Parameters

We create synthetic data that has a underlying model

$$y = \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \beta_3 \cdot x_3 + \beta_0 + \epsilon$$

where

- x_1 is x,
- x_2 is $\sin(x)$,
- x_3 is $(x-5)^2$.

The data contains 50 points and is created with

- $\beta_0 = 5$,
- $\beta_1 = .5$,
- $\beta_2 = .5$,
- $\beta_3 = -.02$.

```
In [ ]: nsample = 50
        sig = 0.5
        x = np.linspace(0, 20, nsample)
        X = np.column_stack((x, np.sin(x), (x-5)**2, np.ones(nsample)))
        beta = [0.5, 0.5, -0.02, 5.]

        y_true = np.dot(X, beta)
        y = y_true + sig * np.random.normal(size=nsample)
```

The first three points are

```
In [ ]: X[:3]
```

```
In [ ]: y[:3]
```

Again, fit the OLS model using the data and print the summary.

```
In [ ]: res = sm.OLS(y, X).fit()
        print(res.summary())
```

Extract other quantities of interest:

```
In [ ]: B = res.params
        yHat = res.predict()
        print('Parameters: ', B)
        s = "y = {0:6.5f} x_0 + {1:6.5f} x_1 + {2:6.5f} x_2 + {3:6.5f} x_3"
        print(s.format(B[0], B[1], B[2], B[3]))
```

Let us look at the errors and compare the original true values and predicted values of the dependent variable.

In [ ]:
```
print('Standard errors: ', res.bse)
v = np.column_stack((y, yHat))
print('True and predicted values: \n', v)
```

Draw a plot to compare the true relationship to OLS predictions. Confidence intervals around the predictions are built using the `wls_prediction_std` command.

In [ ]:
```
prstd, iv_l, iv_u = wls_prediction_std(res)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'o', label="data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res.fittedvalues, 'r--.', label="OLS")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
ax.legend(loc='best');
```

OLS with Dummy Variables

- Dummy (or indicator) variables
  - has a value 0 or 1
  - indicates if β_i has an effect
- Dummy variables are used to represent categorical predictor variables that only have two possible values.

Example

Samples are from three groups with group 0 being the omitted/benchmark category. We want to learn a linear model

$$\hat{y} = \hat{\beta}_1 \cdot x_1 + \hat{\beta}_2 \cdot x_2 + \hat{\beta}_3 \cdot x_3 + \hat{\beta}_0$$

where x_2 and x_3 are the dummy variables

- Generate a synthtic data set

In [ ]:
```
nsample = 50
groups = np.zeros(nsample, int)
groups[20:40] = 1
groups[40:] = 2
#dummy = (groups[:,None] == np.unique(groups)).astype(float)
groups
```

In [ ]:
```
dummy = sm.categorical(groups, drop=True)
dummy
```

In [ ]:
```
x = np.linspace(0, 20, nsample)
# drop reference category
X = np.column_stack((x, dummy[:,1:]))
X = sm.add_constant(X, prepend=False)
X
```

In [ ]:
```
# underlying beta used to generate the data
beta = [1., 3, -3, 10] # B1, B2, B3, B0 of the population
y_true = np.dot(X, beta)
e = np.random.normal(size=nsample)
y = y_true + e
y
```

Inspect the data:

In [ ]:
```
print(X[:5,:])
print(y[:5])
print(groups)
print(dummy[:5,:])
```

- Fit the model and get the model summary:

In [ ]:
```
res2 = sm.OLS(y, X).fit()
print(res2.summary())
```

Draw a plot to compare the true relationship to OLS predictions:

In [ ]:
```
prstd, iv_l, iv_u = wls_prediction_std(res2)

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(x, y, 'o', label="Data")
ax.plot(x, y_true, 'b-', label="True")
ax.plot(x, res2.fittedvalues, 'r--.', label="Predicted")
ax.plot(x, iv_u, 'r--')
ax.plot(x, iv_l, 'r--')
legend = ax.legend(loc="best")
```

Joint Hypothesis Test

Given the regression results of the previous example, we want to determine whether both coefficients for the dummy variables are equal to zero, that is, $\beta_2 = \beta_3 = 0$ or equivalently $R \times \beta = 0$.

F Test

We run an F test with $H_0 : \beta_2 = \beta_3 = 0$. The result leads us to strongly reject the null hypothesis.

```
OLS_Result.f_test(r_matrix): perform f_test assuming r_matrix = 0
```

If the null hypothesis is rejected, β_i in the r_matrix are statistically significantly differ from zero

In [ ]:
```
R = [[0, 1, 0, 0], [0, 0, 1, 0]]
print(np.array(R))
print(res2.f_test(R))
```

- The same test can also be specified using a formula-like syntax (similar to R)

In [ ]:
```
print(res2.f_test("x2 = x3 = 0"))
```

Small Group Effects

- For the previous data set, and given test level 0.01, we must reject the null hypothesis. So, $\beta_2 \neq 0$ and $\beta_3 \neq 0$
- If we consider another dataset from a population where β_2 and β_3 have very small values, the F test will no longer reject the Null hypothesis.

In [ ]:
```
beta = [1., 0.3, -0.0, 10]
y_true = np.dot(X, beta)
y = y_true + np.random.normal(size=nsample)

res3 = sm.OLS(y, X).fit()
```

In [ ]:
```
print(res3.f_test(R))
```

In [ ]:
```
print(res3.f_test("x2 = x3 = 0"))
```

Multicollinearity

- Collinearity is a situation where in a multiple regression model, some predictor variable x_2 is very likely a linear combination of another predictor variable x_1. That is for a significant number of observations i, we have

$$X_{2i} = \lambda_0 + \lambda_1 X_{1i}$$

- If collinearity exits, the coefficient estimates of the multiple regression can be significantly affected if a minor changes is made to model specification.

Statsmodels is using endog and exog as names for the data, the observed variables that are used in an estimation problem. Other names that are often used in different statistical packages or text books are, for example,

endog	exog
y	x
y variable	x variable
left hand side (LHS)	right hand side (RHS)
dependent variable	independent variable
regressand	regressors
outcome	design
response variable	explanatory variable

Example

- Load the longley data set which contains collinearity data
  - 15 data records
- Learn a multiple regression model
- Assess the collinearity of the modle

In [ ]:
```
from statsmodels.datasets.longley import load_pandas
y = load_pandas().endog
X = load_pandas().exog
X = sm.add_constant(X)
print(X)
print(y)
```

Fit and summary:

In [ ]:
```
ols_model = sm.OLS(y, X)
ols_results = ols_model.fit()
print(ols_results.summary())
```

Condition Number

- Compute the condition number.
- If the condition number is big, say 20 or more, the collinearity may be severe.

  Calculation:
  - Normalize the predictor variables to have unit length
  - Compute the correlation matrix for X matrix

In [ ]:
```
norm_x = X.values
for i, name in enumerate(X):
    if name == "const":
        continue
    norm_x[:,i] = X[name]/np.linalg.norm(X[name])
norm_xtx = np.dot(norm_x.T,norm_x)
```

- Find eigen values of the correlation matrix
- Condition number is the square root of the ratio of the biggest to the smallest eigen values.

In [ ]:
```
eigs = np.linalg.eigvals(norm_xtx)
condition_number = np.sqrt(eigs.max() / eigs.min())
print(condition_number)
```

Effect of Dropping an Observation

- Dropping a single observation (data item) can have a dramatic effect on the coefficient estimates
- In the following, drop the last data row, and see the percentage changes for the estimated coefficients

```
In [ ]: ols_results2 = sm.OLS(y.iloc[:14], X.iloc[:14]).fit()
        print(ols_results2.params)
```

```
In [ ]: print("Percentage change %4.2f%%\n"*7 % \
              tuple([i for i in (ols_results2.params - \
                                 ols_results.params)/ols_results.params*100]))
```

We can also look at formal statistics for this such as the DFBETAS -- a standardized measure of how much each coefficient changes when that observation is left out.

```
In [ ]: infl = ols_results.get_influence()
```

In general we may consider DBETAS in absolute value greater than $2/\sqrt{N}$ to be influential observations

```
In [ ]: 2./len(X)**.5
```

```
In [ ]: print(infl.summary_frame().filter(regex="dfb"))
```